Mathematica Components

“Mathematica”

mathematica

Frontend (GUI)

MathLink

math

Kernel (Computation)
## Why I hate the Frontend

<table>
<thead>
<tr>
<th><strong>Frontend:</strong></th>
<th><strong>Kernel:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>![Smiley]</td>
<td>![Smiley]</td>
</tr>
<tr>
<td>Nice formatting</td>
<td>Text interface</td>
</tr>
<tr>
<td>![Meh]</td>
<td>![Meh]</td>
</tr>
<tr>
<td>Documentation</td>
<td>No pretty-printing</td>
</tr>
<tr>
<td>![Smiley]</td>
<td>![Smiley]</td>
</tr>
<tr>
<td>Ease of use</td>
<td>1-to-1 relation to definitions</td>
</tr>
<tr>
<td>![Sad]</td>
<td>![Sad]</td>
</tr>
<tr>
<td>No obvious relation between screen and definitions</td>
<td>Interactive and non-interactive</td>
</tr>
<tr>
<td>![Smiley]</td>
<td>![Smiley]</td>
</tr>
<tr>
<td>Always interactive</td>
<td>Scriptable</td>
</tr>
<tr>
<td>![Sad]</td>
<td>![Sad]</td>
</tr>
<tr>
<td>Slow startup</td>
<td>Fast startup</td>
</tr>
</tbody>
</table>

---

T. Hahn, *Getting most out of Mathematica* – p.3
Plan

- Program smart!
- Parallelize!
- Script! Distribute! Automate!
- Crunch numbers outside Mathematica!

But: don’t overdo it.
If your calculation takes 5 min in total, don’t waste time improving.
Program smart!
List-oriented Programming

Using Mathematica’s list-oriented commands is almost always of advantage in both speed and elegance.

Consider:

\[
\text{tab} = \text{Table}[	ext{Random}[], \{10^7\}];
\]

\[
\text{test1} := \text{Block}[\{\text{sum} = 0\},\]
\[
\text{Do}[\text{sum} += \text{tab}[[\text{i}]], \{\text{i}, \text{Length}[	ext{tab}]\}];\]

\[
\text{sum}
\]

\[
\text{test2} := \text{Apply}[	ext{Plus}, \text{tab}]
\]

Here are the timings:

\[
\text{Timing}[\text{test1}][[1]] \Rightarrow 8.29 \text{ Second}
\]

\[
\text{Timing}[\text{test2}][[1]] \Rightarrow 1.75 \text{ Second}
\]
More Speed Bumps

Consider:

```
tab = Table[Random[], {10^5}];

test1 := Block[{res = {}},
  Do[ AppendTo[res, tab[[i]]], {i, Length[tab]} ];
  res ]

test2 := Block[{res = {}},
  Do[ res = {res, tab[[i]]}, {i, Length[tab]} ];
  Flatten[res] ]
```

The timings:

```
Timing[test1][[1]]  ➔ 19.47 Second
Timing[test2][[1]]  ➔ 0.11 Second
```
Assignments that don’t change the content make no copy but just increase the Reference Count.

\[
\begin{align*}
a &= x & \text{(1)} \\
b &= a & \text{(2)} \\
++b & \rightarrow x + 1 & \text{(1)}
\end{align*}
\]
Reference Count and Speed

test1 := ...  
  ... AppendTo[res, tab[[i]]] ...  
  res

test2 :=  
  ... res = {res, tab[[i]]} ...  
  Flatten[res]

test1 has to re-write the list every time an element is added:  
{}   {1}   {1,2}   {1,2,3}   ...  

test2 does that only once at the end with Flatten:

{}   {{},1}   {{{},1},2}   {{{},1},2},3}   ...  

T. Hahn, Getting most out of Mathematica – p.9
More Programming Wisdom

- Michael Trott
  The Mathematica Guidebook for \{Programming, Graphics, Numerics, Symbolics\} (4 vol)
Parallelize!
Parallel Kernels

Mathematica has built-in support for parallel Kernels:

LaunchKernels[];
ParallelNeeds["mypackage"];

data = << mydata;
ParallelMap[myfunc, data];

Parallel Kernels count toward Sublicenses.
# Sublicenses = 8 \times # interactive Licenses.

MPP: 35 interactive licenses (5k€ each), 288 sublicenses.
Parallel Functions

- **More functions:**
  
  - `ParallelArray`
  - `ParallelEvaluate`
  - `ParallelNeeds`
  - `ParallelSum`
  - `ParallelCombine`
  - `ParallelTable`
  - `ParallelDo`
  - `ParallelProduct`
  - `ParallelTry`
  - `ParallelMap`
  - `ParallelSubmit`
  - `DistributeDefinitions`
  - `DistributeContexts`

- **Automatic parallelization (so-so success):**
  
  - `Parallelize[expr]`

- **‘Intrinsic’ functions (e.g. Simplify) not parallelizable.**

- **Multithreaded computation partially automatic (OMP) for some numerical functions, e.g. Eigensystem.**

- Take care of side-effects of functions.

- Usual concurrency stuff (write to same file, etc.).
Script! Distribute! Automate!
Efficient batch processing with Mathematica:

Put everything into a script, using sh’s Here documents:

```bash
#!/bin/sh
.................. Shell Magic
math << _EOF_
............. start Here document (note the \\)
    << FeynArts'
    << FormCalc'
    top = CreateTopologies[...];
    ...
_EOF_
................... end Here document
```

Everything between "<< \tag" and "tag" goes to Mathematica as if it were typed from the keyboard.

Note the "\" before tag, it makes the shell pass everything literally to Mathematica, without shell substitutions.
Scripting Mathematica

- Everything contained in one compact shell script, even if it involves several Mathematica sessions.
- Can combine with arbitrary shell programming, e.g. can use command-line arguments efficiently:

```sh
#!/bin/sh
math -run "arg1=$1" -run "arg2=$2" ... << END
...
END
```

- Can easily be run in the background, or combined with utilities such as make.

Debugging hint: -x flag makes shell echo every statement,

```sh
#!/bin/sh -x
```
Crunch numbers outside Mathematica!
Code generation

- **Conversion** of Mathematica expression to Fortran/C painless.

- Optimized output can easily run faster than in Mathematica.

- **Showstopper:** Functions not available in Fortran/C, e.g. NDSolve, Zeta. Maybe 3rd-party substitute (GSL, Netlib).

- **Mathematica** has built-in C-code generator, e.g.

  ```mathematica
  myfunc = Compile[{{x}}, x^2 + Sin[x^2]]; Export["myfunc.c", myfunc, "C"]
  ```

  But no standalone code: shared object for use with Mathematica (i.e. also needs license).

- **FormCalc’s** code-generation functions produce optimized standalone code.
Code-generation Functions

FormCalc’s code-generation functions are public and disentangled from the rest of the code. They can be used to write out an arbitrary Mathematica expression as optimized Fortran or C code:

- `handle = OpenCode["file.F"]` opens `file.F` as a Fortran file for writing,
- `WriteExpr[handle, {var -> expr, ...}]` writes out Fortran code which calculates `expr` and stores the result in `var`,
- `Close[handle]` closes the file again.

T. Hahn, Getting most out of Mathematica – p.19
Code generation

Traditionally: Output in Fortran. Code generator is meanwhile rather sophisticated, e.g.

- **Expressions too large** for Fortran are split into parts, as in
  
  ```
  var = part1
  var = var + part2
  ...
  ```

- **High level of optimization**, e.g. common subexpressions are pulled out and computed in temporary variables.

- **Many ancillary functions** make code generation versatile and highly automatable, such that the resulting code needs few or no changes by hand:
  
  `VarDecl, ToDoLoops, IndexIf, FileSplit, ...`

T. Hahn, Getting most out of Mathematica – p.20
Output in C99 makes integration into C/C++ codes easier:

SetLanguage["C"]

Code structured by e.g.

- **Loops and tests handled through macros, e.g.**
  
  \[
  \text{LOOP}(\text{var},1,10,1) \ldots \text{ENDLOOP}(\text{var})
  \]

- **Sectioning by comments, to aid automated substitution e.g. with sed,** e.g.

  \[
  * \text{BEGIN VARDECL} \ldots * \text{END VARDECL}
  \]

- **Introduced data types** RealType and ComplexType for better abstraction, can e.g. be changed to different precision.
The **MathLink API** connects Mathematica with external C/C++ programs (and vice versa). **J/Link** does the same for Java.

```mathematica
:Begin:
:Function:   copysign
:Pattern:    CopySign[x_?NumberQ, s_?NumberQ]
:Arguments:  {N[x], N[s]}
:ArgumentTypes: {Real, Real}
:ReturnType:  Real
:End:

#include "mathlink.h"

double copysign(double x, double s) {
   return (s < 0) ? -fabs(x) : fabs(x);
}

int main(int argc, char **argv) {
   return MLMain(argc, argv);
}

For more details see arXiv:1107.4379.
```